DOI 10.12694/scpe.v19i1.1392
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 1, pp. 39–51. http://www.scpe.org 

# ROUND ROBIN WITH LOAD DEGREE: AN ALGORITHM FOR OPTIMAL CLOUDLET DISCOVERY IN MOBILE CLOUD COMPUTING

RAMASUBBAREDDY SOMULA* AND SASIKALA R†

**Abstract.** Mobile devices have become essential in our daily lives but it has limited resources such as battery life, storage, and processing capacity. Offloading resource intensive task into the cloud is an efficient approach to improve battery utilization, storage capacity and processing capabilities. Efficiently computing using cloud resources to process offloaded task in order to improve response time and reduce both tasks' waiting time and latency problems is one of the main goals in mobile cloud computing (MCC). In order to improve user satisfaction and performance of the mobile application, a cloudlet framework concept has been developed to reduce latency problems which improve response time. The cloudlet brings the cloud closer to the user to perform a computational task. This article proposes a new balancing model among cloudlets in mobile cloud computing environment to find the required resources and create an impact on performance. The efficient load balancing model makes mobile cloud computing more attractive and improves user satisfaction. This paper introduces a Round Robin with Load degree algorithm for public cloudlets in mobile cloud computing using a switch mechanism to choose different approaches for different situations. This algorithm uses game theory based load balancing approach to improve application response time in public mobile cloud environments.

**Key words:** public cloud; cloudlet, offloading, mobile cloud computing, load balancing model, game theory

**AMS subject classifications.** 68M14, 91A80

**1. Introduction.** Mobile cloud computing (MCC) can be viewed as a resource to fill the gap between limited resources of mobile devices (MD) and computation requirement of MD. According to the definition provided by [9], mobile cloud computing is a task in which both computation and storage will happen outside of smart device. The mobile cloud computing has attracted the attention of industries do to its benefits. It reduces the efforts in developing application and also allows users to use the latest technology on demand basis (or) pay per use model. The MCC has three main components: mobile device, wireless connection, cloud infrastructure where both data storage and processing power will be available. Computation offloading means that a computation part of mobile application is offloaded into remote cloud for execution in order to extend mobile device capabilities. Computation offloading can be sufficient when network connection is good and cloud provides enough resources [31]. The cloud provides resource to MD with the help of network communication in heterogeneous environment [22].

The cloudlet is an emerging technology in mobile cloud computing that brings cloud near to mobile user (being like a data center in the box). The cloudlet is based on a cluster of physical machines which allows nearby mobile users to use available resources [24]. It can be available around mobile users. Figure 1.1 depicts the cloudlet architecture: a cloudlet is a kind of small scale server, associated with specific resources. This can be accessed by mobile devices within specific range. The cloudlet provides cloud resources to users in a short time and with a certain level of throughputs [15].

The cloudlet can be composed of three layers: component layer, node layer and cloudlet layer. The component layer provides resources to higher layer through interface overlooked by execution environment. The node can be formed by running a node agent on one or more execution environments on the top of OS. Multiple nodes can be formed as cloudlet by a cloudlet agent [27].

The cloudlet can be efficient and scalable, but maintaining and processing a large number of incoming jobs consistently is very difficult. Therefore, the researchers are paying attention to the load balancing problem. The behavior of the incoming jobs from mobile users is unpredictable and each cloudlet capabilities cannot be constant (number of physical systems can always be changed). In order to improve the system performance and maintain stability, controlling the workload distributed among cloudlets is very important.

Load balancing mechanisms can be classified into two types: static and dynamic [25]. In case of static, the mechanism is simple and does not consider nodes performance before distributing load into heterogeneous environment and handles nodes showing low load variation while dynamic mechanism uses nodes performance

*School of Computer Science and Engineering (SCOPE), VIT University, Vellore, India (svramasubbareddy1219@gmail.com).
†School of Computer Science and Engineering (SCOPE), VIT University, Vellore, India.(sasikala.ra@vit.ac.in)

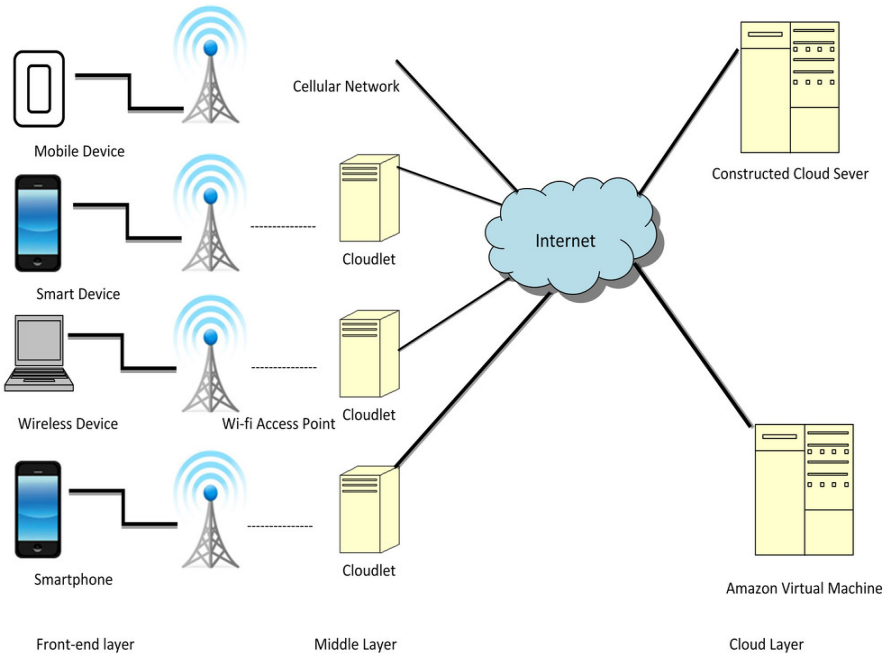


FIG. 1.1. *Cloudlet Architecture*

while distributing work load. Dynamic nature algorithms are more complex than static nature algorithms. The behavior of dynamic scheme is changing according to node status or performance and brings additional cost.

In our model, the proposed method also follows dynamic load balancing as the load is distributed with progression of time. The proposed model contains a cloudlet_ manager cloudlet agents. The cloudlet manager directs the jobs from users to cloudlet. The cloudlet agent collects the information of cloudlet and updates the status in cloudlet manager. The dynamic model is using system status and decides the best balancing strategy. This process will influence other working nodes in cloudlet. The Load balancing model is mainly based on public cloud environment. The cloudlets are distributed in geographical locations. The Cloudlet_manager maintains distributed cloudlets and each cloudlet is assigned to one agent. The Cloudlet_manager will select a cloudlet for incoming jobs and agent decides load balancing strategy. The existing algorithms allocate arriving jobs from mobile users to available cloudlets without knowing the status of the cloudlet. These makes jobs to wait for long time in queue until other jobs get finished. To address this problem a new balancing algorithm is proposed in this paper. The proposed algorithm computes Load_degree of each node in cloudlet to drive the incoming jobs to a particular cloudlet in geographical area.

**2. Background.** Ofloading mobile application into remote cloud to avoid limitation of mobile devices and increases the performance of application was studied in [4, 23, 20, 19]. There are many mechanisms introduced to find when the task should offload on cloudlet [5, 8, 11, 26]. Usually, the model for offloading mobile application into remote cloud consists of components for both client and server. The client components work on mobile devices at client side whereas server components at remote cloud. The client side components monitor network performance, predicts the computational resources of mobile application and estimating time for local as well as remote execution times. By using this information the client components make decisions on how much portion of the application to offload [7, 8]. The task offloading can be done by migrating VM from one cloudlet to another cloudlet [13]. The recent works ThinkAir [17], cloudclone [7], proposed the concept of offloading mobile application from single-machine execution into distributed execution environment automatically. The study of the efficient model for offloading task solution to improve QoS of user application is done in [14]. The users are able to offload tasks into a cloudlet by selection, in the papers [28, 29], [6]. There are studies which are mainly focused to minimize revenue of service providers in cloud using linear programming model for offloading.

The author of [10] proposed to incorporate QoS requirements and energy consumption.

The problem with offloading task to remote cloud is the latency which disturbs user experience with interactive applications. A cloudlet is a new solution for high-latency problem between mobile user and remote cloud. The cloudlet is a small scale datacenter which is available to nearby users. This provides rich computing resource access and improves the performance of mobile application. Generally, the offloading application can be done by the following VM-based approach [10]. The mobile device continues the process of creating VM instances of applications and they are transferred to cloudlet, then the device remotely executes the offloaded task on cloudlet. Once offloaded task execution is done, its result is sent back to the device. This approach can cause the cloud to become constrained if more number of requests increases from the user. There are studies which have presented novel model to find a task with average response time at cloudlet using queuing theory [16]. When a cloudlet goes overloaded, the task is offloaded into remote cloud from current cloudlet. The author of [27] proposed a new model for offloading tasks which can be distributed to multiple cloudlets by dividing application into multiple independent executable components at runtime. The advantage of this approach is the distribution of the workload into multiple cloudlets allowing parallel processing. However, this approach is not applicable at some conditions especially when network condition is poor.

Load balancing is still a new problem in MCC, but load balancing mechanism is not a new concept, it is a widely adapted mechanism in different fields especially in cloud computing. Load balancing concept has been clearly described in the whitepaper [1] which describes the tools commonly utilized for implementing load balancing in cloud. The load balancing scheme is used in mobile cloud computing among widely distributed cloudlets in geographical environment. This scheme is a new approach in MCC to reduce latency, response time and user satisfaction by balancing arriving jobs with a help of agent and cloudlet manager. The cloud can be well utilized in geographical area by dividing the entire cloud into different regions [30], but it requires new approach to improve the user satisfaction.

We propose a new architecture for representing the distributed different cloudlets in geographical environment. Different load balancing algorithms are there since decades. They are Equally Spread Current Execution, Round Robin, Ant colony [2]. Ant colony algorithm can be used to analyze and compare the performance with different algorithms in terms of time and cost [21]. Some of the widely used algorithms are similar to the operating system method allocations, such as the First Come First Serve (FCFS), Shortest Job First (SJF), Round Robin (RR). In our proposed mechanism Round Robin is used for simplicity.

**3. Cloudlet Architecture.** MCC has many models; our model is based on public cloud. The user will get many services from the public cloud infrastructure provided by the cloud service provider. The cloudlet is a subarea in the cloud computing. The public cloud is spread on a large area and hosts a certain number of cloudlets and each cloudlet is composed with many number of nodes. The cloudlets are distributed in geographical areas which are managed by the cloudlet_manager. The following architecture depicts the cloudlet in different regional areas. The load balancing strategy follows by selection of cloudlet in public cloud environment. When job arrives from mobile user, then cloudlet_manager will receive and decide cloudlet for processing the arriving job. If the status of cloudlet (nodes) is normal then the job allocated to the local cloudlet otherwise, the job gets forwarded to another cloudlet. Figure 3.1 depicts the complete idea of the proposed model and process.

**3.1. Cloudlet_Manager and Agent.** The cloudlet_manager maintains information of cloudlets distributed in geographical area. The cloudlet_manager takes the arriving jobs and sends it to the optimized cloudlet. Each cloudlet has an agent which will refresh the system status. Small dataset requires more processing. The information related to the nodes in cloudlet are maintained by the agent. Based on this, the load balancing strategy is decided. Figure 3.3 depicts the association between the agent and cloudlet_manager.

Figure 3.2 describes the flow of cloudlet selection. When jobs arrive at the cloudlet_manager, the first step is to select the resourceful cloudlet. The status of cloudlet can be represented in three models.

1. LCL (low loaded cloudlet): when load of the nodes exceeds to A, turn to LCL status.
2. BCL (Balanced cloudlet): when load of the nodes exceed to B, turns to BCL status.
3. OCL (Overloaded cloudlet): when load of the nodes exceeds to C, turns to OCL status

The parameters A, B, C (threshold values) are set by cloudlet agents. The cloudlet_manager will maintain continuous connection with the agents of different cloudlets. When a job arrives to the cloudlet_manager

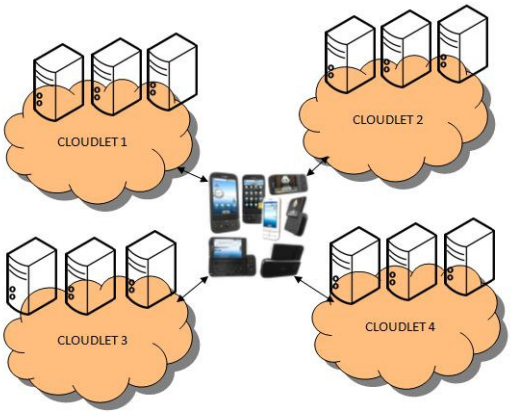


FIG. 3.1. *Cloudlet Selection*

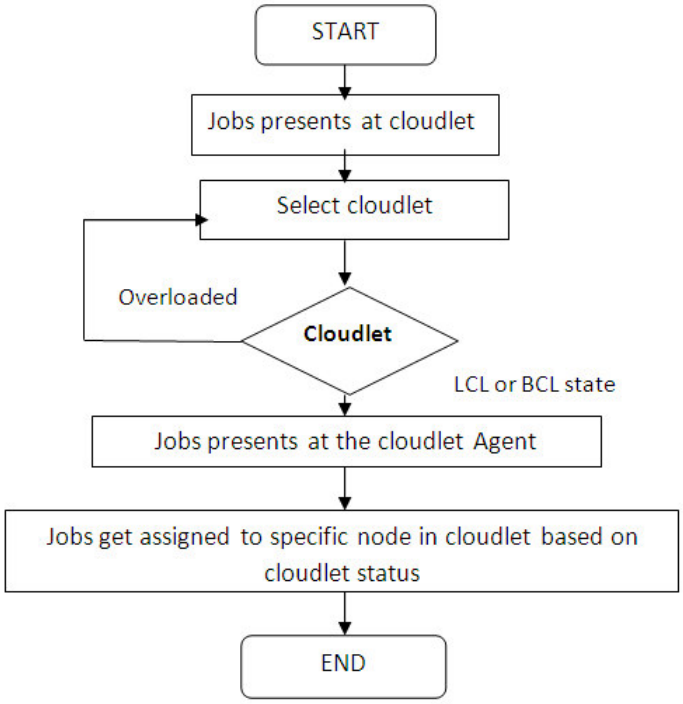


FIG. 3.2. *Flow of Job Scheduling*

questions the cloudlet agent where the job is to be located. If the cloudlet status is LCL or BCL then job is executed locally. Otherwise, another cloudlet is found with status either LCL or BCL. The algorithm is depicted in Algorithm 1.

**3.2. Assigning jobs to nodes in the cloudlet.** The cloudlet agent gathers information about all nodes in cloudlet to calculate the degree of each node. This is important to evaluate each node status in cloudlet selection process. Based on the status of each cloudlet the job gets assigned to it. The first step is to find the degree of nodes in each cloudlet.

The calculation of degree of each node depends on the dynamic and static parameters. The static parameters contain the number of CPUs, CPU speed, size of the memory, software information, etc. The dynamic

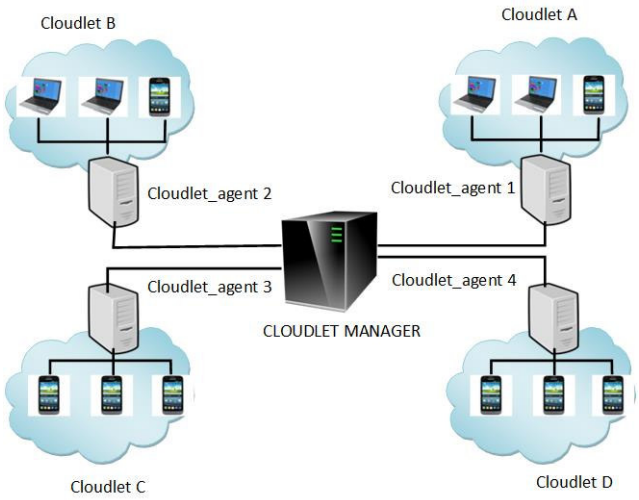


FIG. 3.3. *Cloudlet Management Architecture*

**Algorithm 1** Efficient Cloudlet Finding

```
1: begin
2:     while job do
3:         SearchPerfectCloudlet (job)
4:         if cloudlet_State=LCL || Cloudlet_State=BCL then
5:             Send Job to Cloudlet
6:         else
7:             Search for another Cloudlet;
8:         end if
9:     end while
10: end
```

parameters contain data about usage ratios of CPU, memory, speed, network bandwidth, etc. The evaluation of load degree with different parameters is depicted in the following section.

The cloudlet Load_degree values are input to the cloudlet agents for creating tables. Each cloudlet in mobile cloud computing maintains an agent that decides the status of the cloudlet based on values available in the table. The table updating is done at particular periodic time $T$. When jobs arrives at cloudlet, the cloudlet agent assigns the jobs to a particular node in cloudlet based on cloudlet load status. The cloudlet agent assigns jobs to another cloudlet if cloudlet is not having sufficient resources to handle the incoming jobs.

**4. Cloudlet selection with load balancing algorithm.** A load balancing mechanism is used to improve the performance of mobile application by selecting resourceful cloudlet. There is no efficient load strategy to satisfy all mobile users problems. Each load strategy has its own advantage in specific area. Multiple load balancing methods have been developed to solve the existing problem and improve existing solutions. In our model, the load strategy method is changed according to the system status information. System information can always be changed based on the load balance method that has been adapted. In this model, simple methods can be used for LCL state with complex method for BCL. This model mainly aims to select resourceful cloudlet to reduce waiting time and improve the performance. An enhanced round robin method is used for LCL while BCL uses game theory based load balancing method.

**4.1. Load Balancing for LCL.** When the cloudlet load status is LCL, the nodes in cloudlet are resourceful for handling the arriving jobs. Then the cloudlet processes the arriving jobs as fast as possible and also a simple algorithm is used for scheduling different jobs to nodes. There are many simplest algorithms for

**Algorithm 2** Calculation of Load Degree

**Step 1**: Let the load parameter set $B = \{B_1, B_2, B_3, \ldots, B_m\}$ with each $B_i (1 \leq i \leq m, B \in [0,1])$ the parameter inside set can be static (or) dynamic. M denotes number of parameters.
**Step 2**: Evaluate load degree of node as follows:

$$\text{Load\_degree}(N) = \sum_{i=1}^{m} A_i B_i$$

$A_i (\sum_{i=1}^{N} A_i = 1)$ is representing weights of the jobs, it may differ every time. N represents current cloudlet.
**Step 3**: Calculate the size of the cloudlet from the node degree statistics with the formula

$$Load\_degree_{avg}(N) = \frac{\sum_{i=1}^{N} Load\_degree(N_i)}{n}$$

The threshold Load_degreehigh is applicable for various situations based on value of step 3. Where N is number of nodes in cloudlet.
**Step 4**: The Cloudlet can be classified into three states as follows:
LCL (Low loaded cloudlet): when there is no job to be processed then cloudlet turns to be LCL state.

$$\text{Load\_degree}(N) = 0$$

BCL (Balanced cloudlet): when a cloudlet is neither LCL nor OCL, then cloudlet turns to be BCL state.

$$0 < \text{Load\_degree}(N) = 0 \leq Load\_degree_{high}$$

OCL (Overloaded cloudlet): when cloudlet nodes are busy and unable to process incoming jobs then cloudlet turns to be OCL state

$$Load\_degree_{high} < Load\_degree(N)$$

scheduling arriving jobs such as round robin algorithm, weighted round robin algorithm, dynamic round robin algorithm [18]. The Round robin (RR) scheduling strategy is the simplest algorithm for allocating jobs to the servers. RR will send all incoming jobs to queue inside the server but not the main status of each connection. RR scheduling methods consider every node in cloudlet equally so that equal chance is given for every node in cloudlet to be chosen. However, in a cloudlet each node differs with respect to configuration and performance. RR can be improved by assigning arriving jobs based on load degree of each node. The incoming jobs get stored into the circular queue. The system goes through the queue over and over. Each cloudlet in public cloud maintains a table for the load status of the nodes, based on which the nodes get placed in the table from low load to high load. The order of the nodes in table will be changed based on status of the load status table. However, the inconsistency problem has been taken between read and write at refresh interval $T$. The jobs keep on arriving at cloudlet manger when load balancing table is being refreshed. This makes wrong cloudlet selection and wrong order of cloudlets in the table. Since the table will have old data after getting refreshed, inconsistency problem will lead the arriving jobs to get assigned to wrong node in cloudlet. To address this problem, the cloudlet manager should maintain two load refresh tables: Table 1 and Table 2. A label called read or write can be assigned to each table through flag. The read table contains information of load of the nodes in cloudlet which can be used by improved round robin based on load degree evolution. The write table will be updated with new load information. One table provides exact location of the node in the queue and other table is getting written with new information of the nodes. Both read and write labels will be exchanged to avoid inconsistency problem. Figure 4.1 sketches the inconsistency problem to be addressed by read and write tables.

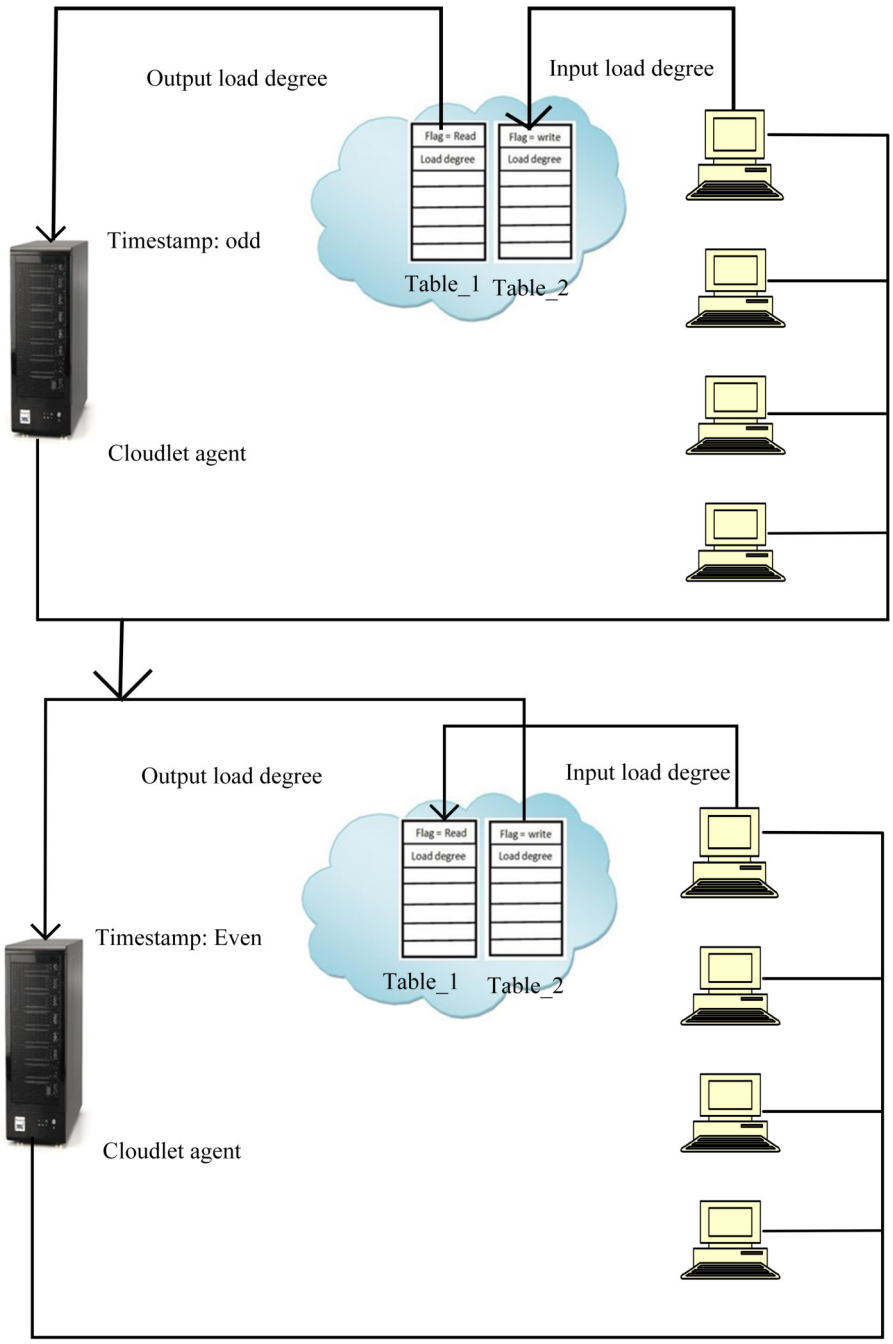


FIG. 4.1. *Efficient model for inconsistency problem*

**4.2. Load Balancing for BCL.** When a cloudlet status is BCL, then all jobs arrives to BCL than LCL, every user wants to finish their work as early as possible. A cloudlet needs a special method to finish all the incoming requests from mobile users. So different strategies are used for improving response time. Static load balancing based on game theory is proposed by chronopulos [20]. This gives us a new idea of cloudlet in mobile cloud computing: the load balancing in mobile cloud computing can be viewed as a game. In the game theory there are cooperative games and non- cooperative games. In a cooperative game, the decision maker will take

decision by discussing with other player or comparing each node with each other. In a non- cooperative game the decision is taken for his own benefit and the other players in the game will take their own decision. The system will reach to Nash equilibrium where each player will make his/her own optimized decision and no player gets benefited by changing his or her own plan, while remaining players decisions are unchanged.

The game theory has been proposed by many studies to solve various problems and was adapted by different areas. It is mainly introduced for distributed computing and later it has been popular in cloud computing too. A dynamic load balancing based on game theory is proposed by Aote and Khara [3]. In this method, users being decision makers in non-cooperative game. Mobile cloud computing is integrated with multiple technologies, networking, mobile computing and cloud computing. Mobile cloud computing is also viewed as distributed computing because this MCC technology is integrated with cloud computing. So the load balancing algorithm can be viewed as a non-cooperative game. The players in the game are nodes and jobs. For instance there are 'n' nodes in current cloudlet with 'm' arriving jobs. The following define this process:

$\beta_i$: Each node processing ability, $i= 1, 2, \ldots, n$

$\varphi_j$: Each job execution time

$\varphi : \sum_{j=1}^{N} \varphi_j$: total time of the entire cloudlet for processing $\varphi < \sum_{i=1}^{n} \beta_i$

$F_{ji}$: Job $j$ get assigned to node $(\sum_{i=1}^{n} F_{ji})$ and $0 \leq F_{ji} \leq 1$

Finding appropriate value $F_{ji}$ is an important step. Grosu et al. [12] has proposed a model called best reply by using this model to evaluate $F_{ji}$ for each node with a greedy algorithm. The procedure will produce a Nash equilibrium to reduce response time of each job. The balancing mechanism changes according to the system status information. Figure 4.2 depicts the proposed model.

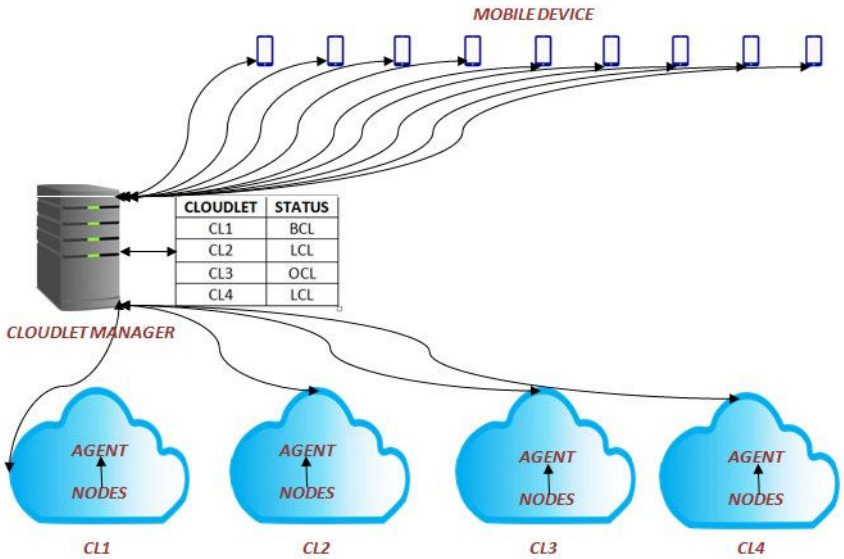


FIG. 4.2. *Proposed cloudlet selection model*

**4.3. Extended Round Robin Algorithm.** We use the Algorithm 3 to find optimal cloudlet to process tasks coming from mobile users in MCC environment.

**5. Performance Evaluation.** In this section we describe how the proposed algorithm performs in various work loaded environments using cloudlets. The computing performance of proposed algorithm is compared with

**Algorithm 3** Extended Round Robin Algorithm

**Step 1**: Calculate load degree of current node based on following equation:

$$\text{Load\_degree}(N) = \sum_{i=1}^{m} A_i B_i$$

**Step 2**: Calculate average load of the cloudlet from each node statistics:

$$Load\_degree_{avg}(N) = \frac{\sum_{i=1}^{N} Load\_degree(N_i)}{n}$$

**Step 3**:Classify the states of cloudlet based on following statistics:
LCL: Load_degree$(N) = 0$
BCL: $0 <$ Load_degree$(N) \leq Load_degree_{high}$
OCL: $Load\_degree_{high} < Load\_degree(N)$
**Step 4**: The job from user will get placed based on load such as LCL,BCL,OCL.

```
1: begin
2:     while job do
3:         SearchPerfectCloudlet (job)
4:         if cloudlet_State=LCL || Cloudlet_State=BCL then
5:             Send Job to Cloudlet
6:         else
7:             Search for another Cloudlet;
8:         end if
9:     end while
10: end
```

TABLE 5.1
*Simulation environment*

| Symbol | Definition | Predefined |
|---|---|---|
| N | Number of cloudlets | 30 |
| $\mu_i$ | Response rate | 20 |
| $\lambda_i$ | Arrival rate at cloudlet | 5 |
| K | Number of servers at cloudlet | 4 |
| A | Number of cloudlet agents | 30 |
| M | Cloudlet Manager | 1 |

an existing algorithm.

**5.1. Experimental results.** We have conducted an experiment by sending a number of requests to different cloudlets. We observed the cloudlet selection process based on load_degree of different cloudlets. If the number of tasks at one cloudlet is more than the cloudlet capacity then the tasks are automatically leaded to another cloudlet by considering cloudlet status without wasting time for cloudlet response.

We use MATLAB environment to generate a random number of cloudlets and requests. Each cloudlet load status evaluated by considering capacity of each node involved in particular cloudlet. Furthermore, the information of load of each cloudlet will be updated with cloudlet_manager. We simulate a cloudlet environment in MATLAB by assuming number of cloudlet (1 to N) and response rate for each cloudlet $\mu_i$ by sampling normal distribution $(6, 3) > 0$, the number of nodes setup in cloudlet is mean of 3 (Table 5.1). In this model, we assumed that the system should not exceed $\mu_i$.

Figure 5.1 shows the comparison between the proposed and the existing algorithm: the proposed algorithm decreases the response time of the cloudlet from 0.5 to 0.9 and the probability of processing a certain number

of requests of the users is higher than for the conventional algorithm.

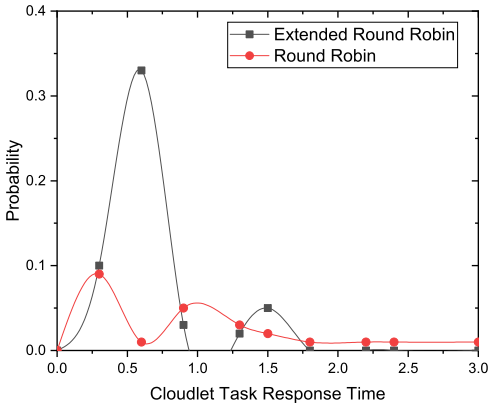


FIG. 5.1. *Probability of Response Time*

Figure 5.2 shows the minimum and maximum task response time of the cloudlets. When the average response time is higher, say R=1.5, the outgoing task demands in the overloaded cloudlets are lesser than the underloaded cloudlets. Also the data flow among the cloudlets and the response time decreases. Similarly, when the average task response time is set low, say R=0.6, the outgoing task demands in the overloaded cloudlets are higher than underloaded cloudlets. The dataflow increases in the cloudlets, thus the response time increases. When the average response time is set to zero, all the cloudlets are fully loaded.

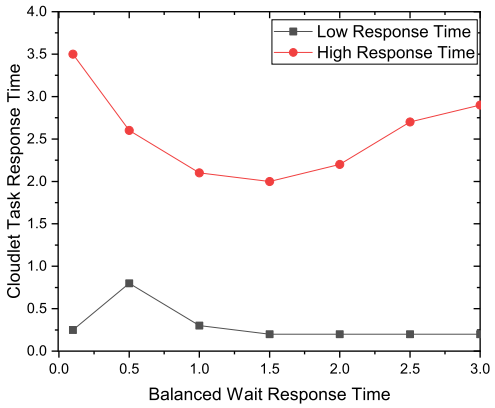


FIG. 5.2. *Low and High Response Time for Average Number of Tasks*

Figure 5.3 states that the proposed algorithm is not affected even though the number of cloudlets increases respectively. Examine the response time of proposed algorithm when there are no requests coming from users as well as requests from users with increasing number of cloudlet. When the count of overloaded cloudlet increases, the task response time also increases.

Figure 5.4 shows the running time of proposed algorithm and existing algorithms (SJF, FCFS, Round Robin) when the number of cloudlets is increased. The time was changed according to the number of cloudlets, system configuration, network status. The graph represents the running time captured from 2.40 GHz Intel Core i5 with 16GB RAM; the number of incoming requested to be processed in system over 0.4 seconds with 400

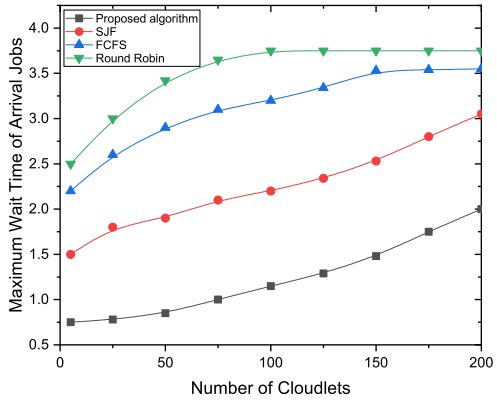


FIG. 5.3. *Delay for Arrival Jobs with Different Cloudlets*

cloudlets. The table updating is done at particular periodic time $T$ when jobs arrives at cloudlet: the cloudlet agent assigns jobs to particular nodes in cloudlet based on cloudlet load status.

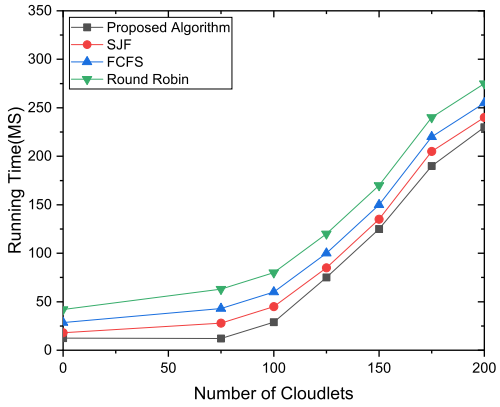


FIG. 5.4. *Running time of proposed algorithm on number of cloudlets*

Figure 5.5 shows the cloudlet task response time with and without the proposed algorithm in a static network. As graph shows, the number of incoming jobs increases, while the task response time also increases under normal distribution. As can be seen, the cloudlet response time can be different from the existing algorithm. The execution time includes transferring time and processing time. If the number of requests comming from the users increases then the performance of existing algorithms is decreasing.

Figure 5.6 shows the cloudlet task response time for proposed method, SJF, FCFS and RR algorithms in a static network. The X-axis represents the cloudlet response time and the y-axis represents the cloudlet arrival rate. As graph shows, the Proposed Method is efficient in comparison with the other three existing algorithms.

**6. Conclusion and Future work.** This paper proposed a novel conceptual framework which is mainly focusing on selection of distributed resourceful cloudlet in a public cloud with the help of a load balancing algorithm. The Cloudlet_manager and Cloudlet_agent are responsible for scheduling the arriving jobs in the

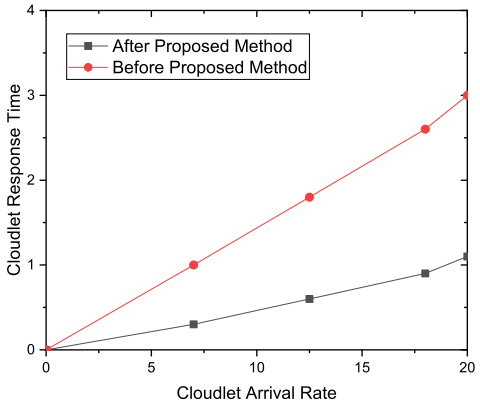


FIG. 5.5. *cloudlet response time for cloudlet arrival rate*

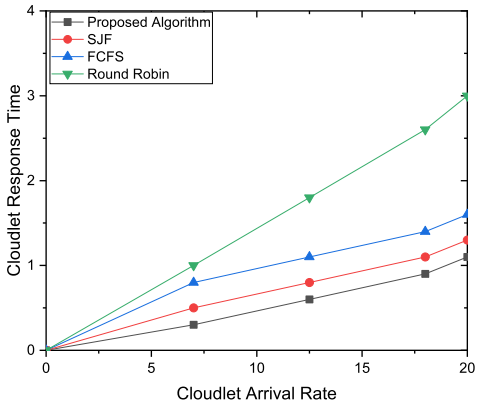


FIG. 5.6. *The response time of cloudlet arrival rate for SJF, FCFS and RR*

cluster of cloudlets. The job scheduling strategy is changed according to the status of the cloudlet of load degree. The proposed algorithm is compared with SJF, FCFS and Round Robin. The results are showing that the response time and the execution time are better than the ones of existing algorithms. Future work need to focus on different aspects, as a Cloudlet selection process is not a simple task: the framework need to be enhanced to represent cloudlet selection process in details, for instance, the number of hosts in cloudlet and how far each host is away from other host, host may be away from other hosts in same cloudlet. The Cloudlet_agent refreshes system information at particular time period.

## REFERENCES

[1] B. ADLER AND S. ARCHITECT, *Load balancing in the cloud: Tools, tips and techniques*, 2012.
[2] M. H. AGHDAM, N. GHASEM-AGHAEE, AND M. E. BASIRI, *Text feature selection using ant colony optimization*, Expert systems with applications, 36 (2009), pp. 6843–6853.
[3] S. S. AOTE AND M. KHARAT, *A game-theoretic model for dynamic load balancing in distributed systems*, in Proceedings of the International Conference on Advances in Computing, Communication and Control, ACM, 2009, pp. 235–238.
[4] R. BALAN, J. FLINN, M. SATYANARAYANAN, S. SINNAMOHIDEEN, AND H.-I. YANG, *The case for cyber foraging*, in Proceedings of the 10th workshop on ACM SIGOPS European workshop, ACM, 2002, pp. 87–92.

[5] N. Bobroff, A. Kochut, and K. Beaty, *Dynamic placement of virtual machines for managing sla violations*, in Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on, IEEE, 2007, pp. 119–128.
[6] V. Cardellini, V. D. N. Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli, *A game-theoretic approach to computation offloading in mobile cloud computing*, Mathematical Programming, 157 (2016), pp. 421–449.
[7] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, *Clonecloud: elastic execution between mobile device and cloud*, in Proceedings of the sixth conference on Computer systems, ACM, 2011, pp. 301–314.
[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, *Maui: making smartphones last longer with code offload*, in Proceedings of the 8th international conference on Mobile systems, applications, and services, ACM, 2010, pp. 49–62.
[9] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, *A survey of mobile cloud computing: architecture, applications, and approaches*, Wireless communications and mobile computing, 13 (2013), pp. 1587–1611.
[10] E. Gelenbe, R. Lent, and M. Douratsos, *Choosing a local or remote cloud*, in Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on, IEEE, 2012, pp. 25–30.
[11] L. Gkatzikis and I. Koutsopoulos, *Migrate or not? exploiting dynamic task migration in mobile cloud computing systems*, IEEE Wireless Communications, 20 (2013), pp. 24–32.
[12] D. Grosu, A. T. Chronopoulos, and M.-Y. Leung, *Load balancing in distributed systems: An approach using cooperative games*, in Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, IEEE, 2001, pp. 10–pp.
[13] J. Heo, K. Terada, M. Toyama, S. Kurumatani, and E. Y. Chen, *User demand prediction from application usage pattern in virtual smartphone*, in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 449–455.
[14] D. T. Hoang, D. Niyato, and P. Wang, *Optimal admission control policy for mobile cloud computing hotspot with cloudlet*, in Wireless Communications and Networking Conference (WCNC), 2012 IEEE, IEEE, 2012, pp. 3145–3149.
[15] Y. Jararweh, F. Ababneh, A. Khreishah, F. Dosari, et al., *Scalable cloudlet-based mobile computing model*, Procedia Computer Science, 34 (2014), pp. 434–441.
[16] L. Kleinrock, *Queueing systems, volume 2: Computer applications*, vol. 66, wiley New York, 1976.
[17] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, *Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading*, in Infocom, 2012 Proceedings IEEE, IEEE, 2012, pp. 945–953.
[18] D. MacVittie, *Intro to load balancing for developersthe algorithms*, 2012.
[19] N. Palaniappan and S. Ramasamy, *A survey on procedures dealing with mobile offloading schemes*, 7 (2017), p. 178.
[20] S. Penmatsa and A. T. Chronopoulos, *Game-theoretic static load balancing for distributed systems*, Journal of Parallel and Distributed Computing, 71 (2011), pp. 537–555.
[21] M. Randles, D. Lamb, and A. Taleb-Bendiab, *A comparative study into distributed load balancing algorithms for cloud computing*, in Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, IEEE, 2010, pp. 551–556.
[22] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, *Heterogeneity in mobile cloud computing: taxonomy and open challenges*, IEEE Communications Surveys & Tutorials, 16 (2014), pp. 369–392.
[23] M. Satyanarayanan, *Pervasive computing: Vision and challenges*, IEEE Personal communications, 8 (2001), pp. 10–17.
[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, *The case for vm-based cloudlets in mobile computing*, IEEE pervasive Computing, 8 (2009).
[25] N. G. Shivaratri, P. Krueger, and M. Singhal, *Load distributing for locally distributed systems*, Computer, 25 (1992), pp. 33–44.
[26] H. N. Van, F. D. Tran, and J.-M. Menaud, *Sla-aware virtual resource management for cloud infrastructures*, in Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on, vol. 1, IEEE, 2009, pp. 357–362.
[27] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, *Cloudlets: Bringing the cloud to the mobile user*, in Proceedings of the third ACM workshop on Mobile cloud computing and services, ACM, 2012, pp. 29–36.
[28] Q. Xia, W. Liang, and W. Xu, *Throughput maximization for online request admissions in mobile cloudlets*, in Local Computer Networks (LCN), 2013 IEEE 38th Conference on, IEEE, 2013, pp. 589–596.
[29] Q. Xia, W. Liang, Z. Xu, and B. Zhou, *Online algorithms for location-aware task offloading in two-tiered mobile cloud environments*, in Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on, IEEE, 2014, pp. 109–116.
[30] G. Xu, J. Pang, and X. Fu, *A load balancing model based on cloud partitioning for the public cloud*, Tsinghua Science and Technology, 18 (2013), pp. 34–39.
[31] K. Yang, S. Ou, and H.-H. Chen, *On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications*, IEEE communications magazine, 46 (2008).

*Edited by:* Dana Petcu
*Received:* Sep 23, 2017
*Accepted:* Feb 16, 2018